ParFlow++: a C++ Parallel Application for Wave Propagation Simulation

Frédéric Guidec, Pierre Kuonen, Patrice Calégari

Paper submitted for the 20th SPEEDUP meeting, to be held in Geneva (CERN) on September 20th, 1996.

ParFlow++: a C++ Parallel Application for Wave Propagation Simulation

Frédéric Guidec, Pierre Kuonen, Patrice Calégari
Department of Computer Science
Swiss Federal Institute of Technology
Email: {guidec,kuonen,calegari}@di.epfl.ch

Abstract

The ParFlow method permits to simulate outdoor wave propagation in urban environment. It compares with the so-called Lattice Boltzman Model (LBM), which describes a physical system in terms of the motion of fictitious microscopic particles over a lattice. One of the objectives of the European project STORMS[1] (Software Tools for the Optimization of Resources in Mobile Systems) is to develop a software tool to be used for design and planning of the UMTS network[2]. This paper gives an overview of the ParFlow method, and reports the design and the implementation of ParFlow++, a parallel object-oriented software for outdoor wave propagation prediction in urban environment. ParFlow++ is being developed in C++, and the resulting code is primarily targeted at MIMD-DM platforms. Although the development of ParFlow++ is still in progress in the STORMS project, a prototype version is available, that compiles and runs on a network of Unix workstations, as well as on the Cray T3D. Experiments achieved with this prototype software lead to promising results, which are discussed in this paper.

Keywords

Wave propagation, simulation, mobile telecommunication, cellular network, Transmission Line Matrix, Lattice Boltzman Model, ParFlow, parallel programming, object-oriented programming.

1. Introduction

Outdoor wave propagation prediction is of great interest to telecommunication operators. Due two the fast growth of radio networks, it becomes important to achieve computer-based simulations of wave propagation. This is particularly true in the context of mobile phone networks, also called “cellular networks”, for it permits to predict the shapes of the cells of any future network. One of the objectives of the European project STORMS (Software Tools for the Optimization of Resourc-

1.STORMS is a European project founded by the European Community and by the Swiss government (OFES).
2.UMTS: Universal Mobile Telecommunication System.

es in Mobile Systems) is to produce pieces of software for wave propagation simulation, in both urban and rural environments. It is planned to use parallel computing to speed up the execution of these pieces of software.

In 1995, a new approach to modelling wave propagation in urban environments based on a Transmission Line Matrix (TLM) was designed by Chopard, Luthi and Wagen [Lut95]. The ParFlow method compares with the so-called Lattice Boltzman Model (LBM), that describes a physical system in terms of the motion of fictitious microscopic particles over a lattice. It permits a bidimensional wave propagation simulation, using a digital city map, and assuming infinite building height. The method can be "naturally" extended to tridimensional wave propagation simulation.

2. The ParFlow method

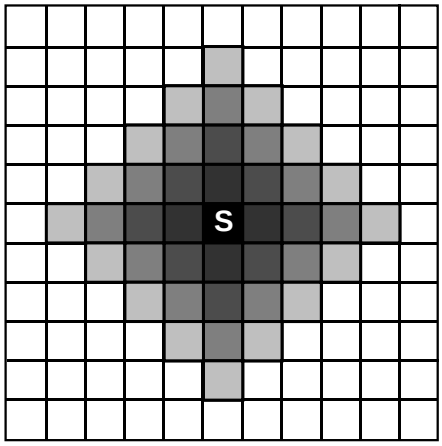

Figure 1: Wave propagation simulation using the ParFlow method

According to the Huygens principle, a wave front consists of a number of spherical wavelets emitted by secondary radiators. The ParFlow method is based on a discrete formulation of this principle. Space can be represented in terms of finite elementary units on a grid. Because Lattice Boltzman models are characterized by a simultaneous independent dynamics and by a very simple numerical scheme, the ParFlow algorithm uses a cellular automaton approach. At each step of the computation, every grid point is updated based on the values of its four neighbours. The computation proceeds until a stable state is reached, or until a predefined number of iteration steps has been run through.

Figure 1 shows the state of the propagation after four iteration steps. The black square marked by a "S" is the location of the radiating source (the emitter). Grey levels indicate at which iteration step a certain grid point was reached by the wave. The darker the square, the earlier the corresponding grid point was reached. One can distinguish between three types of grid points in Figure 1: those that have not been reached by the wave yet, those that have just been reached by the wave (during the current iteration step), and those that were reached during former iteration steps. In any case, points that have been reached by the wave are referred to as *active* points in the remaining of this document, whereas points that have not been reached yet are said to be *inactive*.

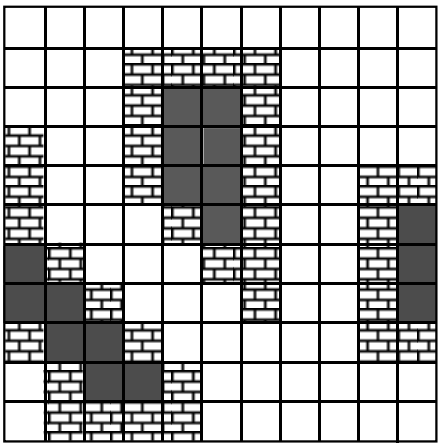

Figure 2: Modelling of obstacles

Obstacles (mainly buildings) are modelled by two kinds of grid points: wall points, and indoor points (see Figure 2). As, in the current ParFlow model, we assume that waves do not penetrate buildings, indoor grid points are not involved in the computation. As for wall points, they are considered as partially absorbent and partially reflecting.

The ParFlow method uses flows instead of fields. Flow values are defined on the edges connecting neighbouring grid points. The flows entering one grid point at time t are scattered at time $t+\Delta t$ among the four neighbouring points, according to the following TLM expression:

$$\begin{bmatrix} f_1(x+\Delta r, y, t+\Delta t) \\ f_2(x-\Delta r, y, t+\Delta t) \\ f_3(x, y-\Delta r, t+\Delta t) \\ f_4(x, y+\Delta r, t+\Delta t) \end{bmatrix} = \frac{1}{2} \cdot \begin{bmatrix} 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} f_1(x, y, t) \\ f_2(x, y, t) \\ f_3(x, y, t) \\ f_4(x, y, t) \end{bmatrix}$$

Due to the discretization of time, it is convenient to distinguish between the flows coming in a grid point, and those going out of this point. For each point, there are thus four incoming flows, and four outgoing flows, as shown in Figure 3.

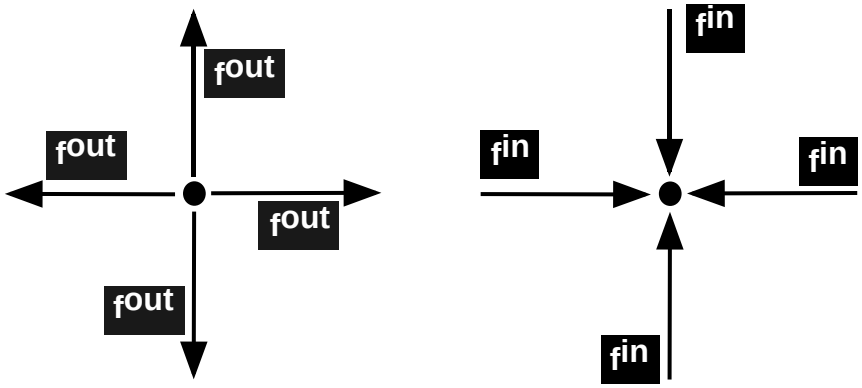

Figure 3: Flows entering and leaving a grid point

The general structure of the algorithm is shown in pseudo-code in Figure 4. For each grid point, incoming and outgoing flows obey the following rules:

- outgoing flows at time t are a linear combination of incoming flows at that time;
- an incoming flow at time t corresponds to the outgoing flow calculated on a neighbouring grid point at time t-Δt;

```
foreach iteration step do
   foreach point in the grid do
      compute outgoing flows based on incoming flows ;
      update incoming flows based on neighbours' outgoing flows ;
   endfor
endfor
```

Figure 4: the ParFlow algorithm (in pseudo-code)

- the field value at time *t* is the sum of the four incoming flows at that time.

These rules apply for all outdoor grid points but the source, which does not propagate incoming flows in the current version of the ParFlow method. By changing the excitation function of the source point, one can simulate different kinds of input signals, such as a sinusoidal signal, a triangular pulse, an impulse, etc. As for wall points, the reflection coefficient and the matrix elements can be modified depending on the kind of wall considered [Lut95].

The ParFlow method must simulate the (theoretically) infinite propagation of a wave on a finite grid. In order to prevent any kind of perturbation along the borders of the grid, one uses one or several layers of special points, whose role is to emulate the attenuation of the wave in open-air. These points are quite similar to traditional outdoor points, except that their reflection coefficient is smaller.

3. Design and implementation of ParFlow++

3.1. Motivations

The grid structure on which the ParFlow algorithm operates, and the way each point of the grid is updated iteratively based on its four neighbours, suggest a very regular implementation. A language primarily targeted at array-based computation, such as Fortran, is perfectly suited for this kind of implementation.

Another advantage of the ParFlow method is that, although the calculations made during each iteration step are theoretically synchronous, updates of points require independent computation. The ParFlow algorithm is therefore a good candidate for parallel implementation.

The first parallel version of the ParFlow algorithm was implemented on a CM2 (Connection Machine) supercomputer, whose SIMD architecture (Single Instruction Multiple Data) provides thousands of synchronous processors [Lut95]. The ParFlow algorithm can be readily and efficiently implemented on SIMD platforms, because it takes advantage of the regular grid data structure and of the synchronous progress of the computation. However, the main disadvantage of such an implementation is its lack of scalability. Since each point of the grid must be allocated to one processing element, the size of the grid is constrained by the size of the parallel machine. Moreover, since the grid structure of the ParFlow algorithm is directly mapped on the topology of the CM2, many processing units model points located inside buildings. These processing units remain idle throughout the entire computation.

Since the current version of ParFlow method does not simulate wave propaga-

tion through buildings, it can be interesting not to model indoor points. Indeed, experience shows that when modelling an urban area, buildings can represent up to 30 % of the surface considered. Not modelling these points permits to avoid a waste of memory space and, as a consequence, of computational power. Such an approach unfortunately leads to an irregular data structure, that makes the implementation much more complicated. In that case, the implementation can take advantage of a programming model that provides powerful features to manipulate irregular data structures. Object-oriented languages are good candidates for this kind of development.

The use of object-oriented programming is not very common in supercomputing. Implementing the ParFlow algorithm using object-oriented techniques thus appears to us as an appealing challenge.

3.2. Parallel, object-oriented implementation of the ParFlow method

ParFlow++ denotes a C++based parallel implementation of the ParFlow algorithm, targeted at MIMD-DM platforms[3]. The ParFlow++ software is intended to be used in the STORMS project to compute cells covered by Base Transceiver Stations (BTSs) in an urban environment.

The platforms primarily targeted in the STORMS project are the Cray T3D supercomputer, and networks of Unix workstations. However, the code of the ParFlow++ software is being developed so as to be easily portable on any other MIMD-DM platform. The object-oriented approach brings several advantages with this respect. The communication facilities required for the parallel implementation of ParFlow++ were encapsulated in one class, whose interface was determined once and for all based on the needs of the application itself. This class may have different implementations depending on any new target platform considered (Cray T3E, Intel Paragon, IBM SP2, etc.), but its interface will remain unaltered whatever this platform.

Due to the amount of outdoor points that must be considered in a simulation (typically, several thousands of points for a single city district), an appropriate policy must be chosen to allocate each point to a processing element of the target platform. In order to obtain good load balancing, ParFlow++ relies on a static data distribution, based on a partitioning of the grid in thin bands. Each band is allocated to a given processor. This kind of partitioning has several advantages:

- The width of each band, and the number of them assigned to each processor, can be adjusted so as to allow a fair distribution of the workload among the processors.
- Allocating thin bands near the source ensures that all processors get their share of work early after the wave started propagating.
- As adjacent bands can be allocated to adjacent processors, communications are only required between neighbouring processors. This characteristic is almost useless when running on a network of workstations interconnected through a traditional Ethernet trunk. On the other hand, on a platform such as the Cray T3D, it permits to prevent, or at least reduce, the well-known contention phenomenon.

3.MIMD-DM: Multiple Instructions, Multiple Dataflow, Distributed Memory.

3.3. Software engineering

ParFlow++ is being developed in an object-oriented way, and according to the fundamental principles of software engineering. The preliminary analysis and design of the software were achieved using the Fusion method [Col95]. The main classes and relationships that were identified during the analysis phase constitute an object model, that is reproduced in Figure 5.

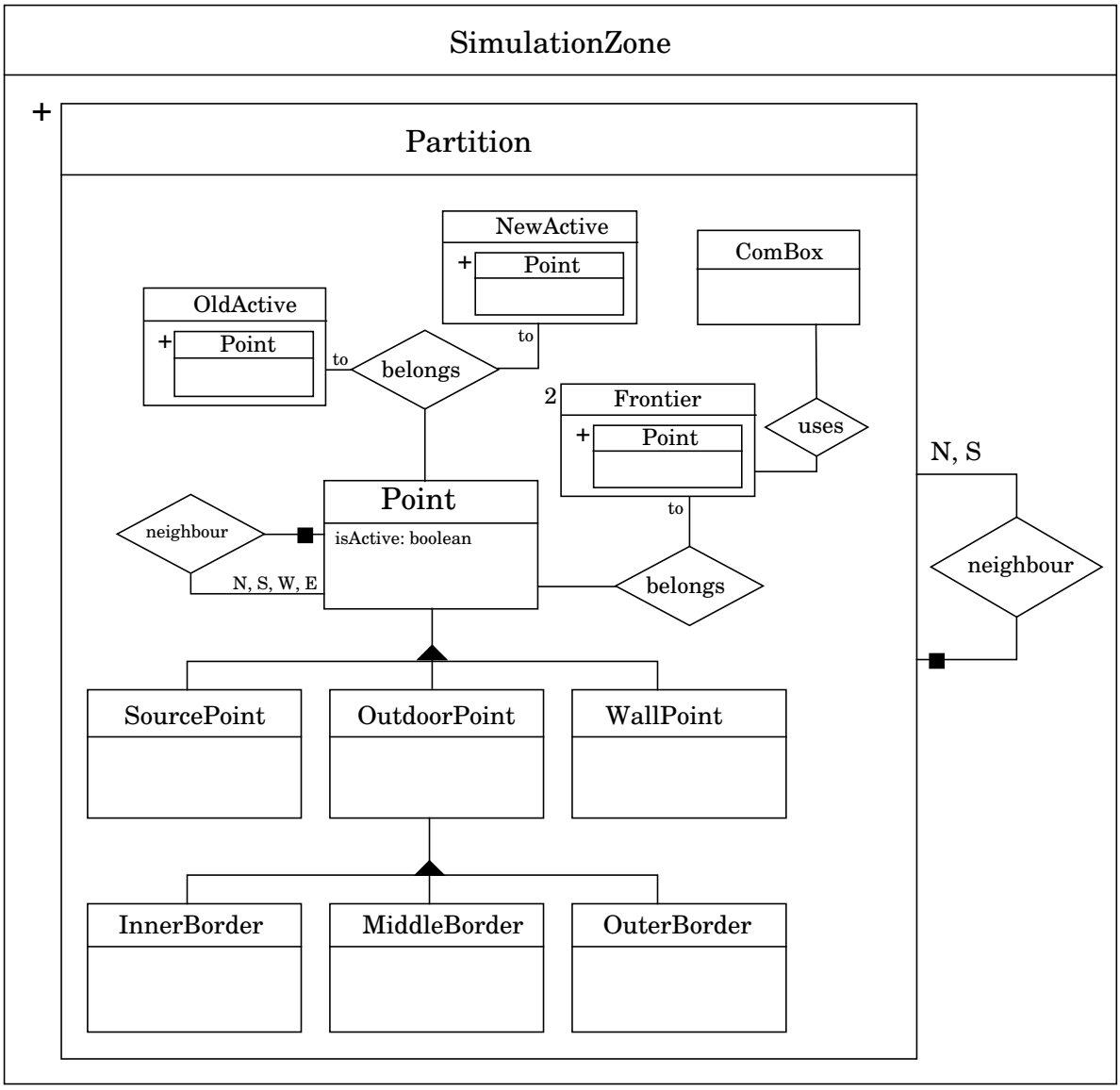

Figure 5: Fusion object model of the ParFlow++ software

The area on which a wave propagation simulation must be achieved is modelled as a simulation zone (instance of the class SimulationZone shown in Figure 5), that is split into partitions[4] (instances of class Partition). Each partition is to be allocated to a given processor. It consists of a collection of instances of the class Point. A

4.Not splitting the simulation zone comes down to running ParFlow++ with a single partition. ParFlow++ then behaves almost like a purely sequential application (the overhead due to the “parallel” code is negligible).

point is the smallest entity that is modelled with ParFlow++. It corresponds to an elementary unit of the grid discussed in Section 2. Since several kinds of points must be considered in this grid (source points, open-air points, wall points, border points, etc.), the object model reproduced in Figure 5 shows that the class Point has several descendants (the black triangle is a symbol for inheritance in the Fusion diagrammatic syntax). These descendant classes all share the same interface, but their implementation may differ slightly. For example, instances of OutdoorPoint, of SourcePoint, and of WallPoint, do not have the same propagation characteristics. Likewise, ParFlow++ implements three layers of border points, that differ mainly in their reflection coefficient.

As explained in Section 2, a point is either active or inactive, depending on whether it has already been reached by the wave whose propagation is being simulated, or not. During each step of the simulation, field values need only be calculated for active points, and the propagation of outgoing flows is only required from active points to their neighbours. Based on these observations, a partition manages several structures internally, in order to reduce the amount of computation performed during one simulation step. Every newly activated point is inserted in a list *newActive*. This list permits to propagate efficiently the activity status after each computation step: the only points that are in a position to be activated are those that are neighbours to members of the *newActive* list, and that are still inactive. Iterating through members of the *newActive* list thus facilitates the identification of new active points. Once a member of *newActive* has activated its neighbours, it is transferred in the list *oldActive*. Hence, it will never be considered again when looking for new active points, although it will still participate in the computation and in the propagation of field values. The parallel version of the ParFlow algorithm is reproduced in pseudo-code in Figure 6.

Besides the two lists that permit to distinguish between “new” and “old” active points, a Partition also manages up to two instances of class Frontier (see Figure 5). A Frontier maintains references to those points that are either on the northern edge, or on the southern edge, of a partition. These points differ slightly from other points of the local partition, for they must interact with neighbouring points that are managed by remote processors. The class Frontier thus ensures the propagation of flows between a partition and one of its neighbours. It also ensures the propagation of the activity status, since a newly active point on an edge of a partition may activate a point in the neighbouring partition. To achieve these goals, the implementation of Frontier relies on the communication facilities offered by the class ComBox.

C++ provides no support for parallel computation. The class ComBox (Communication Toolbox) was developed in order to fill this gap, and bring communication facilities in the object-oriented world of the ParFlow++ software. The interface of this class provides the basic mechanisms needed for sending and receiving messages, for synchronizing several tasks in a SPMD[5] application, etc. The current implementation of ComBox is based on the basic communication services offered by the PVM library [Gei94]. Yet, alternative implementations may be considered in the future. We may, for example, implement the class ComBox using the MPI library, or the POM library. On the Cray T3D, we may use the fast communication func-

5.SPMD: Single Program, Multiple Dataflow.

```
newActive = {source point} ;
oldActive = ∅ ;
foreach iteration step do
   foreach local partition do
      foreach point ∈ oldActive do
         compute outgoing flows based on incoming flows ;
         update incoming flows based on neighbours' outgoing flows ;
      endfor

      oldActive = oldActive ∪ newActive ;
      tmpActive = newActive ;
      newActive = ∅ ;
      foreach point ∈ tmpActive do
         compute outgoing flows based on incoming flows ;
         update incoming flows based on neighbours' outgoing flows ;
         activate not-yet-activated neighbours ;
         newActive = newActive ∪ {newly activated neighbours} ;
      endfor

      send northern and southern frontiers to neighbouring partitions ;
      receive northern and southern frontiers from neighbouring partitions ;
      newActive = newActive ∪ {new active points found in one of the frontiers} ;
   endfor
endfor
```

Figure 6: the ParFlow++ parallel algorithm (in pseudo-code)

tions shmem_get() and shmem_put(). In any case, since the implementation of the ComBox class can be modified without any alteration of its interface, we ensure that the ParFlow++ software is as independent as possible from any kind of target platform.

4. State of the art and preliminary results

A sequential version of the ParFlow++ software was delivered to the STORMS project leaders in August 1996. Although the implementation of the parallel version of the software is still in progress, a first prototype version is now available. This version compiles and runs on a network of Unix workstations, and on the Cray T3D.

The prototype version of ParFlow++ was tested on the Cray T3D. Figure 7 shows the propagation we observed when running a 800 steps simulation on a 500x500 points simulation zone corresponding to a district of the city of Geneva.

For each machine size (number of processors), we tested different partitioning policies (the number of partitions was always a multiple of the number of processing elements). Figure 8 shows, for each machine size, the *best* speedup we observed against the number of processors implied in the computation. When calculating the speedup values shown in this figure, we used as reference times the execution

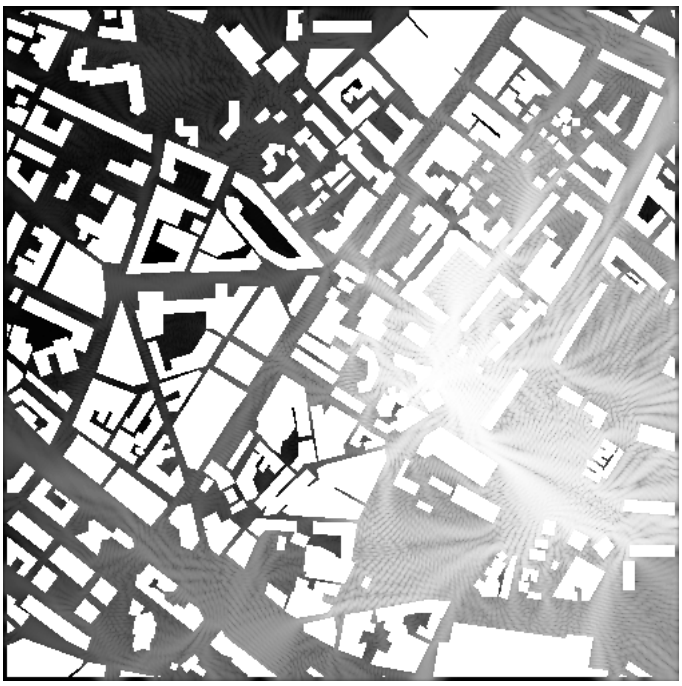

Figure 7: Wave propagation simulation in a district of the city of Geneva

times observed on one processor, with a single partition. Experience shows that, in such conditions, the overhead due to the "parallel" code remains negligible.

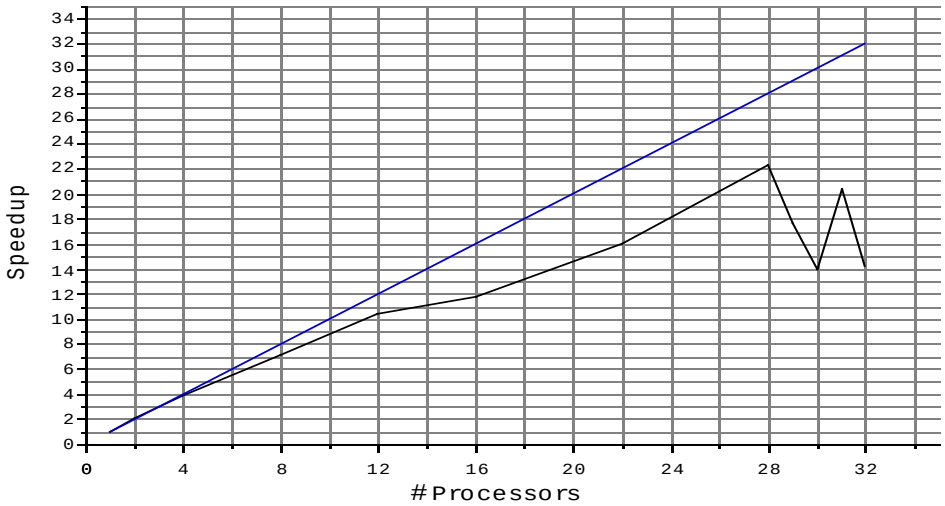

Figure 8: Speedups observed on the Cray T3D

In Figure 8, speedups increase almost steadily with the number of processors, up to 28 processors. The curve then falls down in an unexpected and, to date, still unexplained way. However, considering that ParFlow++ is still in a prototype stage, we think that these results are quite promising, and we feel confident that future releases of ParFlow++ shall exhibit better speedups for many processors. To achieve this goal, thorough profiling and fine tuning of the code are required. Communications can also be improved in many ways (transmission of smaller messages, non-deterministic receives, use of the fast shmem_get() and shmem_put()

primitives on the Cray T3D, etc.).

Figure 9 shows the efficiency observed against the number of partitions, when running the same simulation as that described earlier on 4 processors of the Cray T3D.

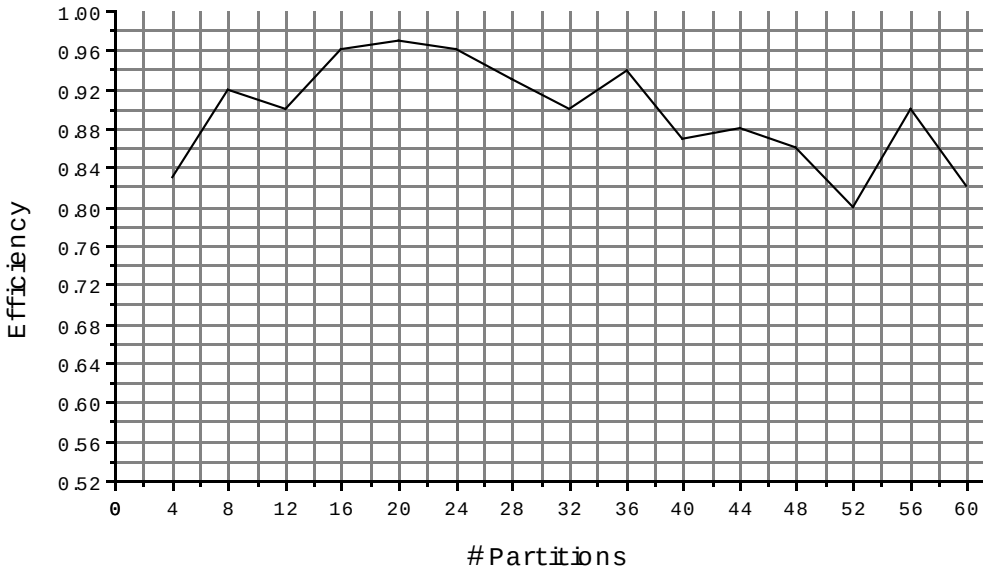

Figure 9: Efficiency observed on the Cray T3D

Figure 9 shows that the best efficiency is obtained when splitting the simulation zone in 20 partitions, which are then fairly distributed over the 4 processors available. These results confirm that the number of partitions (and, as a consequence, their size) strongly influences the behaviour of ParFlow++. Further experiments should permit to understand better how performances depend on the number, on the dimensions, and on the distribution of partitions over the available processors. ParFlow++ currently implements a homogeneous partitioning policy (all partitions have the same size). A heterogeneous partitioning policy, permitting to allocate thinner bands near the source point, should allow a better load balancing of the parallel application.

5. Future work

The performances of the prototype version of the ParFlow++ software are most promising. Yet, ParFlow++ can still be improved in may ways.

During a ParFlow++ simulation, just like during the execution of any other object-oriented program, many objects are created and deleted at runtime. Dynamic memory management is thus a critical issue (especially since no automatic garbage collector is available in a C++ runtime). A thorough profiling of the source code should permit to improve the performances of ParFlow++ significantly.

The communication load can also be reduced, as discussed in Section 4, notably by using faster communication primitives on the Cray T3D, and by exchanging smaller messages. Finally, the analysis of executions using a visualization tool such as Paragraph [Hea91] should help achieve a fine tuning of the ParFlow++ application. It may for example help choose an appropriate partitioning policy, what-

ever the target MIMD-DM platform.

References

[Col95] D. Coleman & al. Object-Oriented Development - *The Fusion Method*. Prentice Hall Object-Oriented Series, Englewood Cliffs, NJ, 1995. ISBN 0-13-338823-9.

[Lut95] P.O. Luthi, B. Chopard and J.-F. Wagen, *Wave Propagation in Urban Micro-cells: a Massively Parallel Approach using the TLM Method*, In Proceeding of PARA'95, Workshop on Applied Parallel Scientific Computing, Copenhagen, August 1995. Also in COST 231 TD(95) 33.

[Gei94] A. Geist et al., *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*, The MIT Press, 1994.

[Hea91] M. Heath and J. Etheridge, *Visualizing the Performances of Parallel Programs*, IEEE Software, Vol. 8, No. 5, Sept. 1991, pp. 29-39.

[Sni96] M. Snir and al., *MPI: The Complete Reference*, Scientific and Engineering Computation Series, The MIT Press, ISBN 0-262-69184-1, 1996.

[Gui96] F. Guidec and Y. Mahéo, *POM: a Parallel Observable Machine," in "Parallel Computing: State of the Art and Perspectives*, Advances in Parallel Computing, Vol. 11, Elsevier, North-Holland publisher, ISBN 0-444-82490-1, 1996.